```
warning off

%functions gradient_descent and newton_iteration take in a starting X
%value and the A value. The user should also provide the function, as well
% as the gradient (and hessian if applicable) as functions at the bottom
% of the program.
%the initial guesses and A value are listed above each function call for
%all four of the tests.



x = [-1.2;1];
A = 100;
gradient_descent(x, A);


x = [-1.2;1];
A = 100;
newton_method(x, A);


x = [-1.2;1];
A = 1;
gradient_descent(x, A);


x = [-1.2;1];
A = 1;
newton_method(x, A);



function gradient_descent(x, A)

    figure;
    hold on;
    xlim([-10 90]);
    xlabel('Number of Iterations');
    ylabel(['Value of the norm of the gradient']);
    if(A == 100)
        title('Value of Norm of Gradient vs. Number of Iterations, A = 100,
 Gradient Descent');
        ylim([0 55]);
    end
    if(A == 1)
        title('Value of Norm of Gradient vs. Number of Iterations, A = 1,
 Gradient Descent');
        ylim([0 5]);
    end

    storage = zeros(2, 6000); %storage for path plot
    c = 0.01;
```

```
rho = .5;
f=func(x, A);
g=grad(x, A);
k = 0; % k = # iterations
funcEval=1; % funcEval = # function eval.


% Begin method
while ( norm(g) > 1e-3 )
    if(norm(g) > 1e-2) plot(k, norm(g), '.'); end
    pk = -g; % steepest descent direction
    a = 1;
    newf = func(x + a*pk, A);
    funcEval = funcEval+1;
    while (newf > f + c*a*g'*pk)
        a = a*rho;
        newf = func(x + a*pk, A);
        funcEval = funcEval+1;
    end

    x = x + a*pk; % gradient descent
    storage(1, k+1) = x(1);
    storage(2, k+1) = x(2);

    f=newf;
    g=grad(x, A);
    k = k + 1;
end
celltable = cell(1,3);
celltable{1,1} = transpose(x);
celltable{1,2} = k;
celltable{1,3} = funcEval;

if(A == 100)
    T = cell2table(celltable,...
    "VariableNames",["Estimate, Gradient Descent, A = 100" "#
Iterations" "# Function Calls"]);
end
if(A == 1)
    T = cell2table(celltable,...
    "VariableNames",["Estimate, Gradient Descent, A = 1" "# Iterations" "#
Function Calls"]);
end
disp(' ');
disp(T);

figure
hold on


if(A == 100)
    title('Path of Optimization, A = 100, Gradient Descent');
    xlim([-1.25, 1.25]);
    ylim([0.85, 1.1]);
```

```
    end
    if(A == 1)
        title('Path of Optimization, A = 1, Gradient Descent');
        xlim([0, 1.5]);
        ylim([0.65, 1.25]);
    end


    contour = @(x,y) 100*(y-x^2)^2 + (1-x)^2;
    fcontour(contour, 'MeshDensity', 200)

    plot(storage(1, 1:k-1), storage(2, 1:k-1), 'kx-');
    plot(storage(1, k), storage(2, k), 'ro', 'MarkerFaceColor', 'r');

    figure
    hold on
    if(A == 100)
        title('Evolution of Cost Function, A = 100, Gradient Descent');
        xlabel('Number of Iterations');
        ylabel('Value of Cost Function');
        xlim([0, 100]);
        ylim([-0.25, 5.5]);
    end
    if(A == 1)
        title('Evolution of Cost Function, A = 1, Gradient Descent');
        xlabel('Number of Iterations');
        ylabel('Value of Cost Function');
        xlim([0, 70]);
        ylim([-0.1, 2]);
    end
    for i = 1:k
        x = [storage(1, i); storage(2, i)];
        plot(i,  func(x, A), '.');
    end


end



function newton_method(x, A)

    figure;
    hold on;
    xlim([0 90]);
    xlabel('Number of Iterations');
    ylabel(['Value of the norm of the gradient']);
    if(A == 100)
        title('Value of Norm of Gradient vs. Number of Iterations, A = 100,
 Newton Method');
        ylim([-0.5 90]);
    end
    if(A == 1)
```

```
        title('Value of Norm of Gradient vs. Number of Iterations, A = 1,
Newton Method');
        ylim([-0.5 10]);
        xlim([0 65]);
    end

    storage = zeros(2, 200); %storage for path plot
    c = 0.01;
    rho = .5;
    f=func(x, A);
    g=grad(x, A);
    h = hessian(x, A);
    k = 0; % k = # iterations
    funcEval=1; % funcEval = # function eval.


    % Begin method
    while ( norm(g) > 1e-3 )
        if(norm(g) > 1e-2) plot(k, norm(g), '.'); end
        pk = -1*(h \ g); % steepest descent direction
        a = 0.1;
        newf = func(x + a*pk, A);
        funcEval = funcEval+1;
        while (newf > f + c*a*g'*pk)
            a = a*rho;
            newf = func(x + a*pk, A);
            funcEval = funcEval+1;
        end

        x = x - a*(h \ g); % newton method
        storage(1, k+1) = x(1);
        storage(2, k+1) = x(2);

        h = hessian(x, A);
        f=newf;
        g=grad(x, A);
        k = k + 1;
    end


    celltable = cell(1,3);
    celltable{1,1} = transpose(x);
    celltable{1,2} = k;
    celltable{1,3} = funcEval;

    if(A == 100)
        T = cell2table(celltable,...
        "VariableNames",["Estimate, Newton Method, A = 100" "# Iterations" "#
Function Calls"]);
    end
    if(A == 1)
        T = cell2table(celltable,...
        "VariableNames",["Estimate, Newton Method, A = 1" "# Iterations" "#
Function Calls"]);
```

```
    end
    disp(' ');
    disp(T);


    figure
    hold on


    if(A == 100)
        title('Path of Optimization, A = 100, Newton Method');
        xlim([-1.5, 1.5]);
        ylim([-0.2, 1.4]);
    end
    if(A == 1)
        title('Path of Optimization, A = 1, Newton Method');
        xlim([-1.5, 1.5]);
        ylim([-0.3, 1.25]);
    end


    contour = @(x,y) 100*(y-x^2)^2 + (1-x)^2;
    fcontour(contour, 'MeshDensity', 200)

    plot(storage(1, 1:k-1), storage(2, 1:k-1), 'kx-');
    plot(storage(1, k), storage(2, k), 'ro', 'MarkerFaceColor', 'r');


    figure
    hold on
    if(A == 100)
        title('Evolution of Cost Function, A = 100, Newton Method');
        xlabel('Number of Iterations');
        ylabel('Value of Cost Function');
        xlim([0, 100]);
        ylim([0, 22]);
    end
    if(A == 1)
        title('Evolution of Cost Function, A = 1, Newton Method');
        xlabel('Number of Iterations');
        ylabel('Value of Cost Function');
        xlim([0, 40]);
        ylim([0, 5]);
    end
    for i = 1:k
        x = [storage(1, i); storage(2, i)];
        plot(i,  func(x, A), '.');
    end

end
```

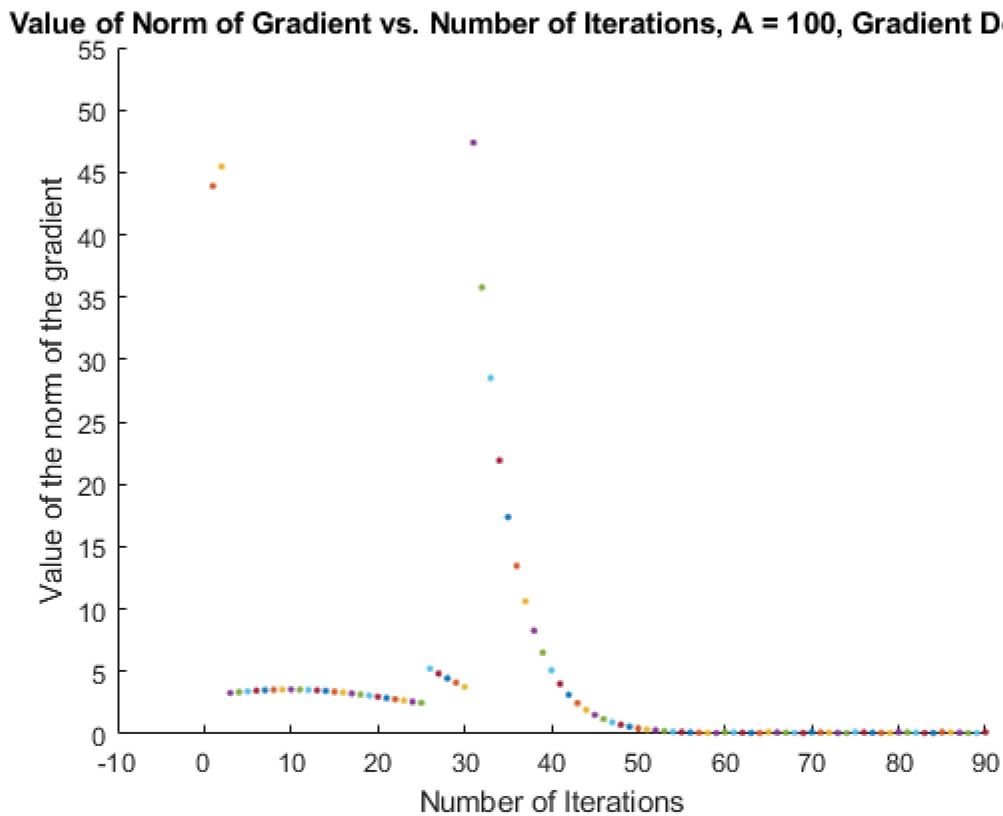
```
%user supplied functions
function y = func(x, A)
    y = A*(x(1)^2 - x(2))^2 + (x(1)-1)^2;
end

function y = grad(x, A)
    y(1) = A*(2*(x(1)^2-x(2))*2*x(1)) + 2*(x(1)-1);
    y(2) = A*(-2*(x(1)^2-x(2)));
    y = y';
end

function y = hessian(x, A)
    y(1,1) = (12*A) * x(1)^2 - (4*A) * x(2) + 2;
    y(1,2) = -(4*A) * x(1);
    y(2,1) = -(4*A) * x(1);
    y(2,2) = (2*A);
end
```

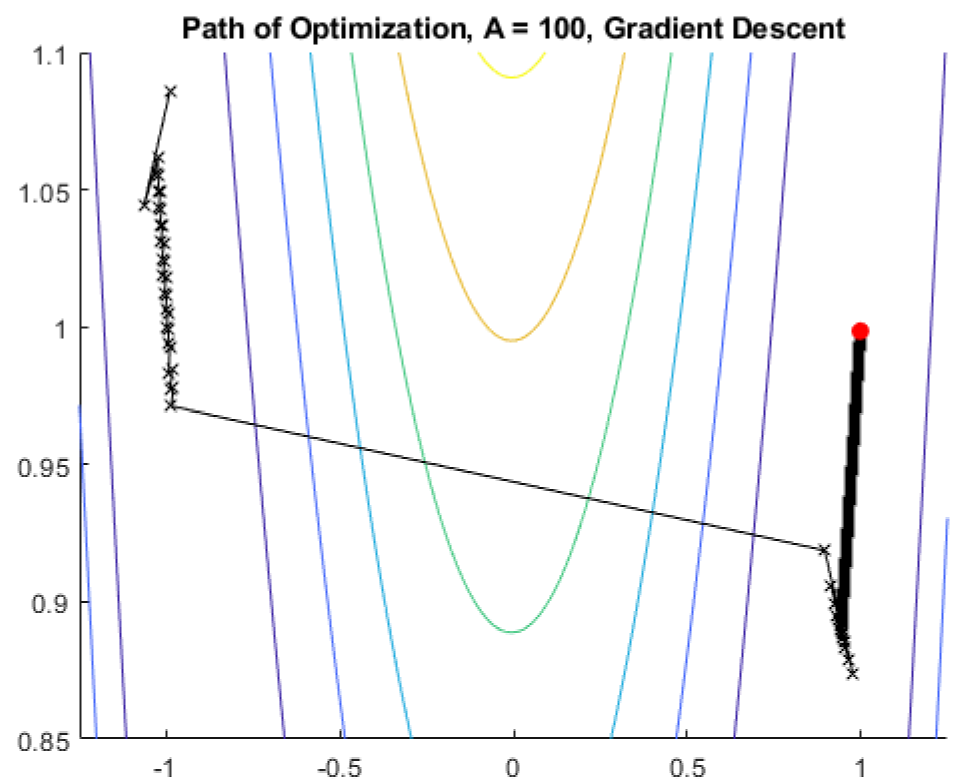
Estimate, Gradient Descent, A = 100	# Iterations	# Function Calls
0.99922 0.99844	5047	50093

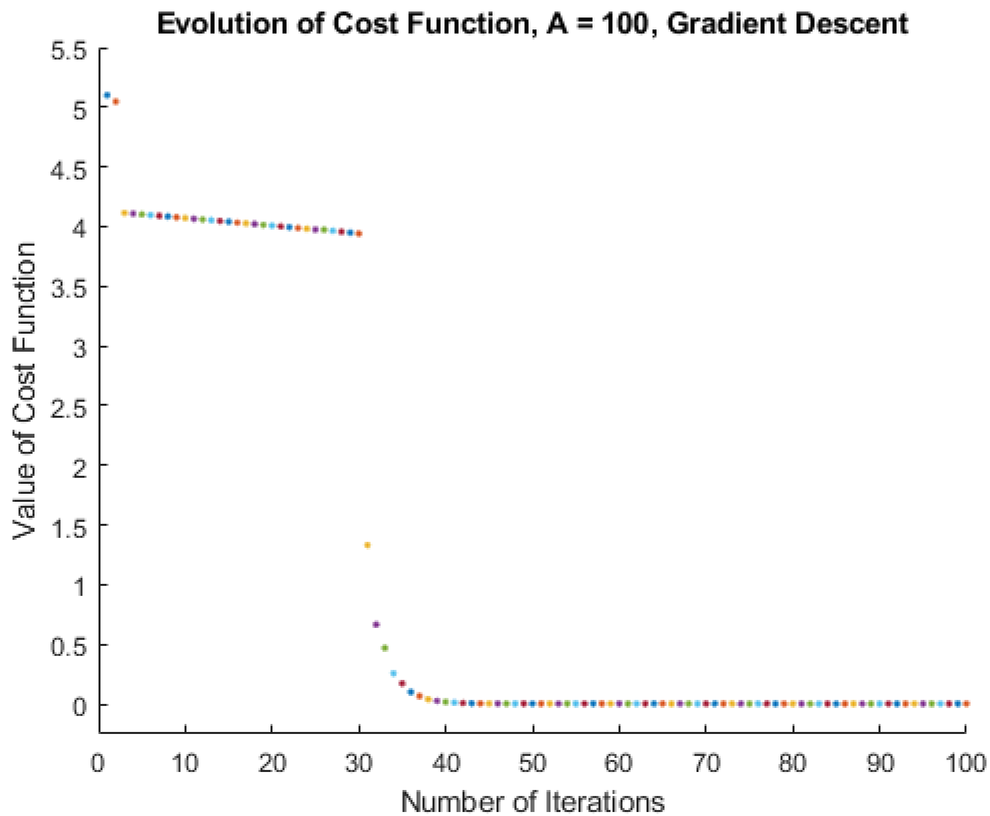
Estimate, Newton Method, A = 100	# Iterations	# Function Calls
0.99986 0.99971	167	168

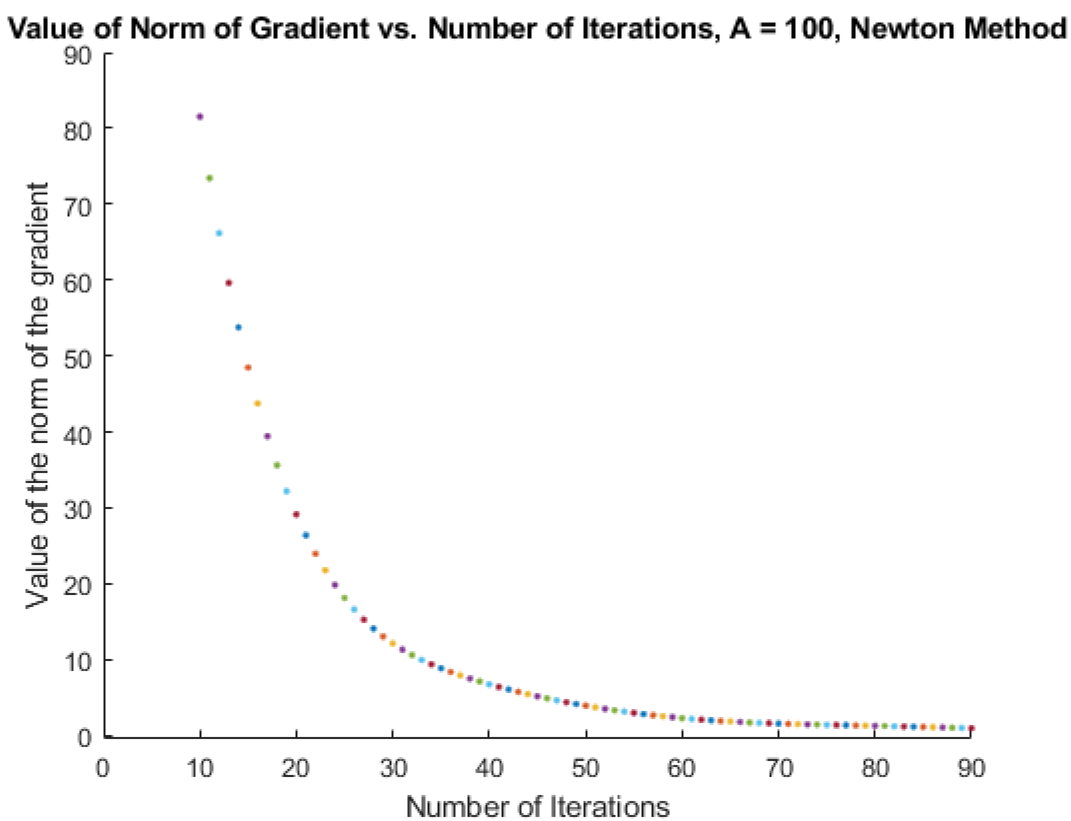
Estimate, Gradient Descent, A = 1	# Iterations	# Function Calls
0.99916 0.99785	66	229

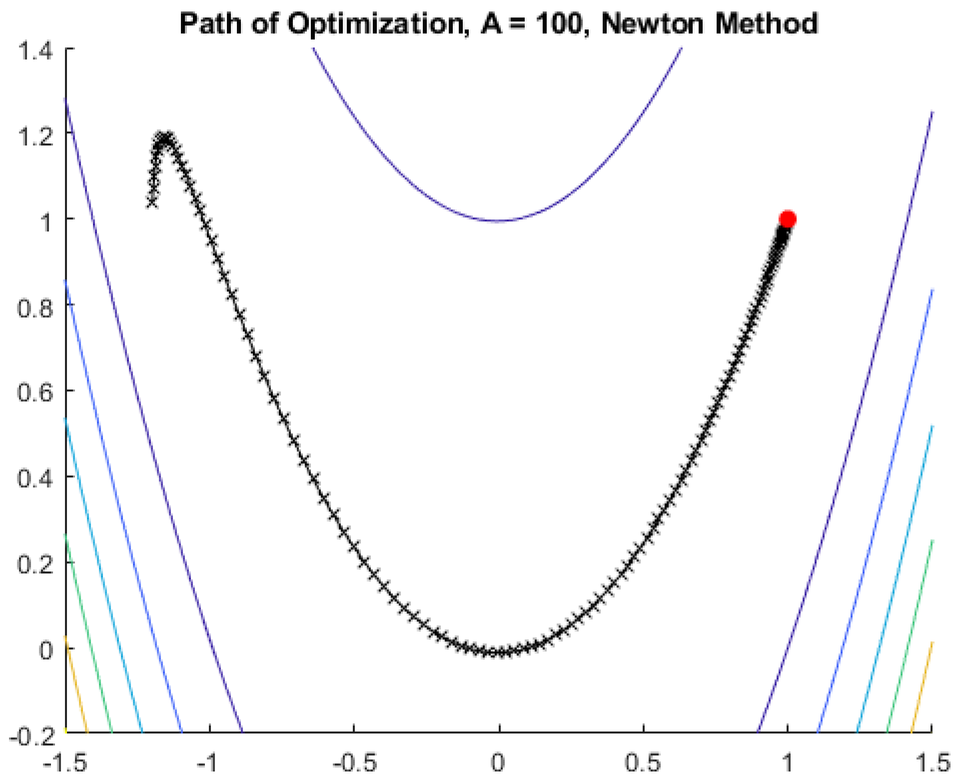
Estimate, Newton Method, A = 1	# Iterations	# Function Calls
0.99929 0.99844	85	86

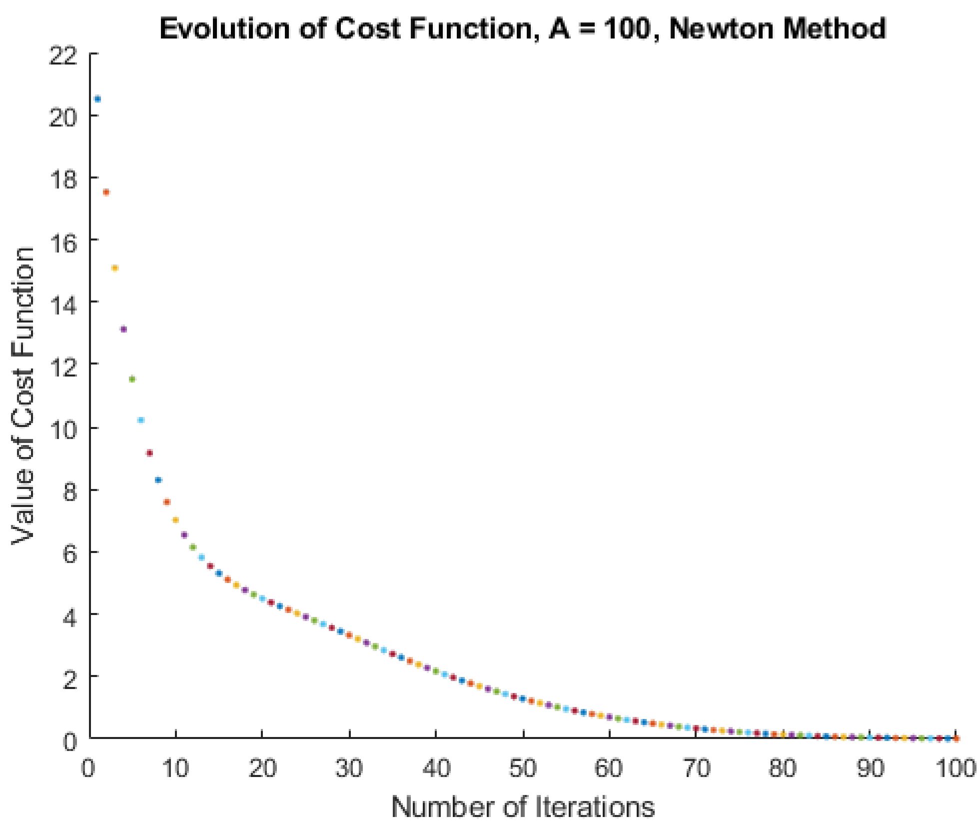

Value of Norm of Gradient vs. Number of Iterations, A = 100, Gradient Descent
Value of the norm of the gradient
Number of Iterations
55
50
45
40
35
30
25
20
15
10
5
0
-10
0
10
20
30
40
50
60
70
80
90

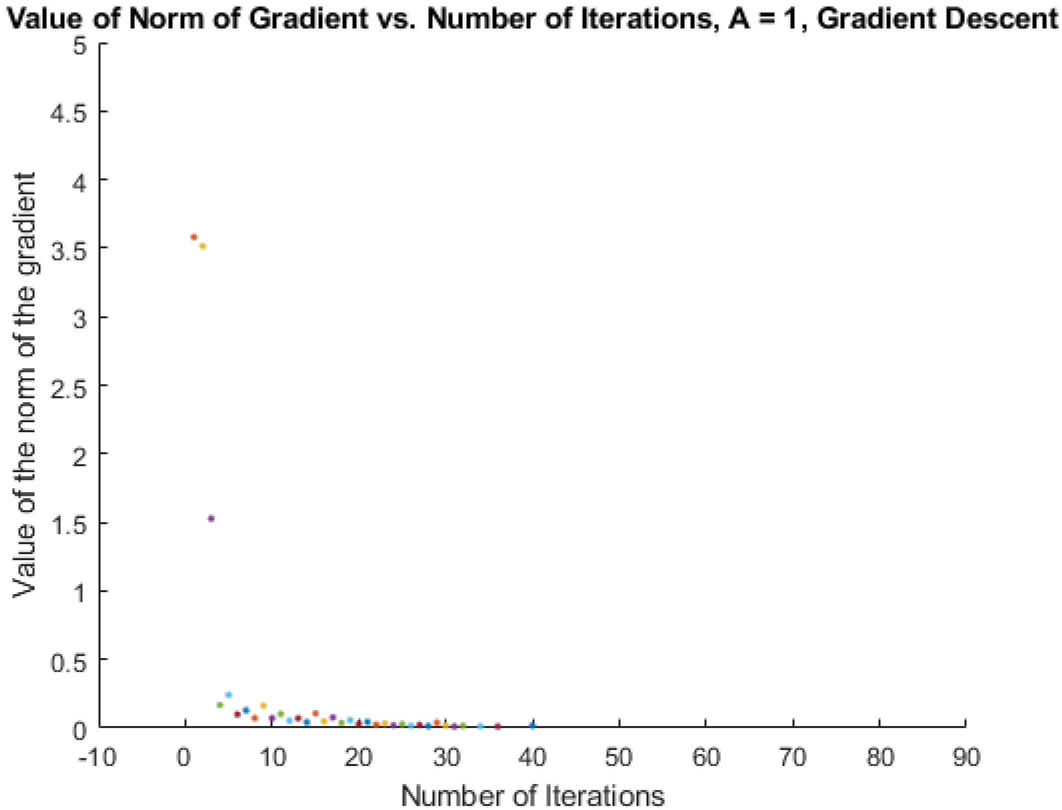

Path of Optimization, A = 100, Gradient Descent
1.1
1.05
1
0.95
0.9
0.85
-1
-0.5
0
0.5
1

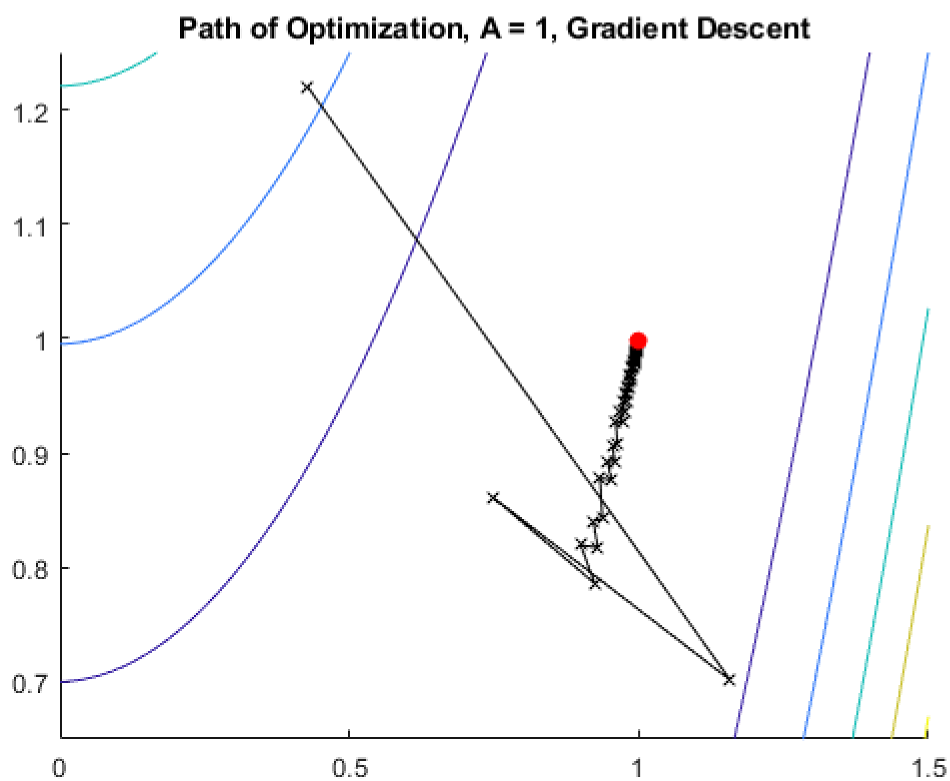

Evolution of Cost Function, A = 100, Gradient Descent
Value of Cost Function
Number of Iterations
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
5
5.5
0
10
20
30
40
50
60
70
80
90
100

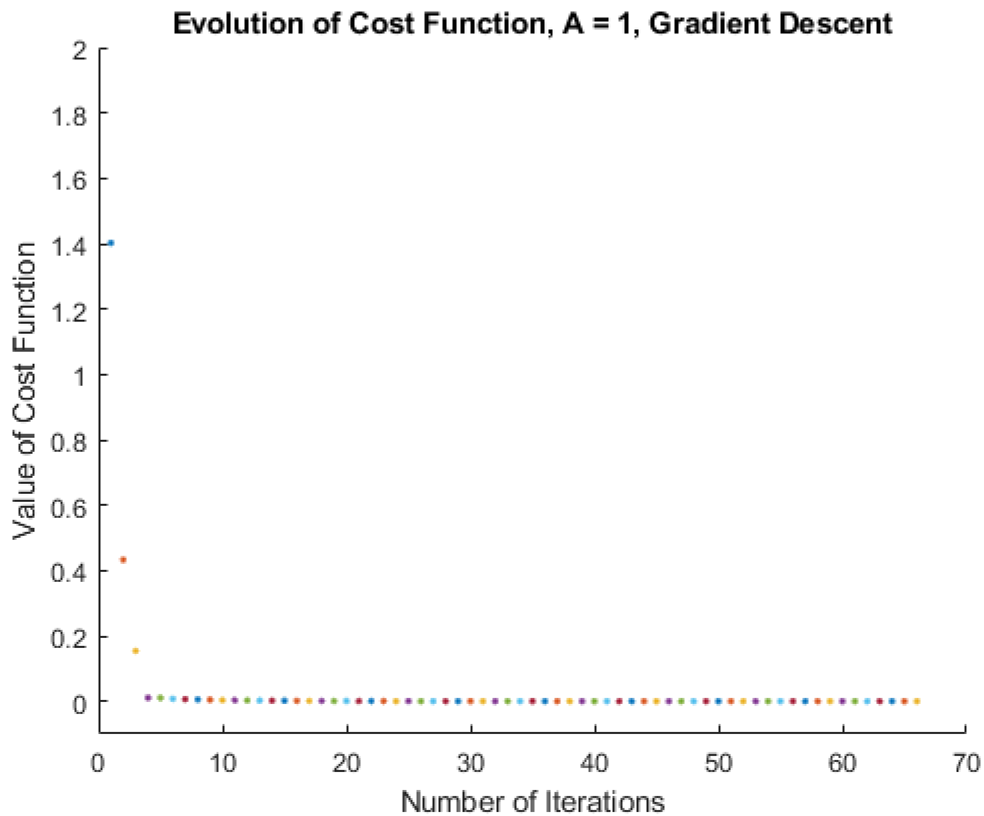

Value of Norm of Gradient vs. Number of Iterations, A = 100, Newton Method
Value of the norm of the gradient
Number of Iterations
0
10
20
30
40
50
60
70
80
90
0
10
20
30
40
50
60
70
80
90


Path of Optimization, A = 100, Newton Method
1.4
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-1.5
-1
-0.5
0
0.5
1
1.5

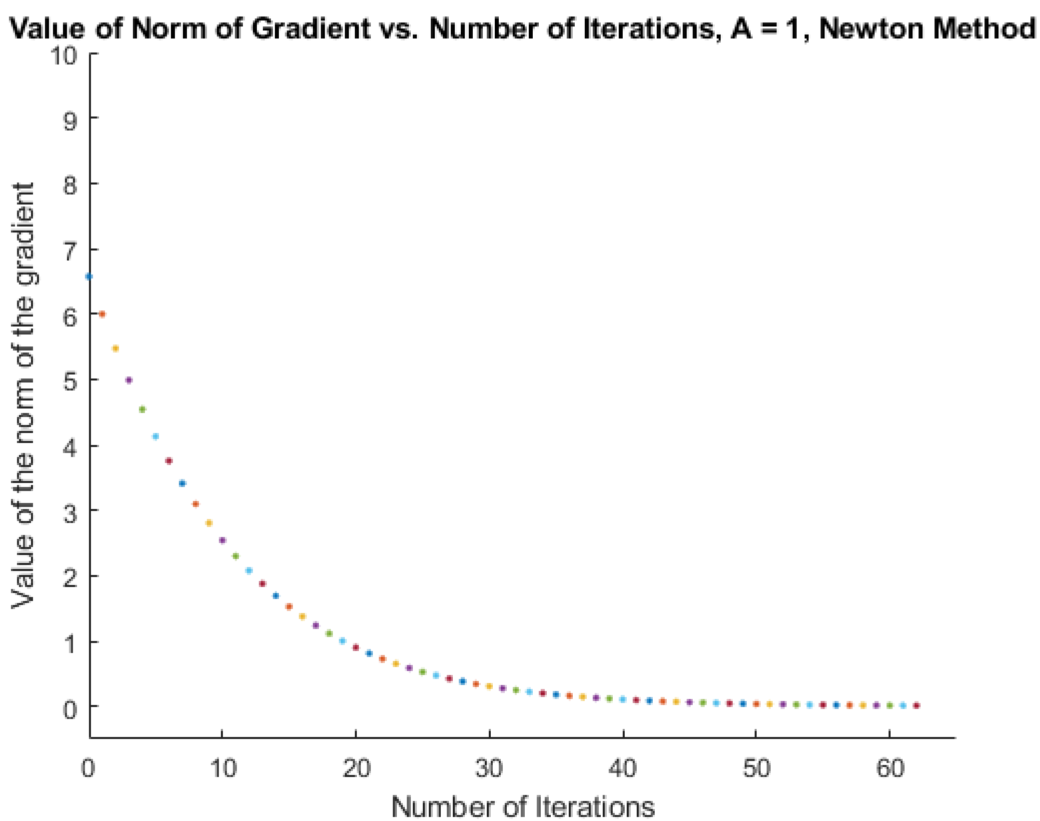

Evolution of Cost Function, A = 100, Newton Method
Value of Cost Function
22
20
18
16
14
12
10
8
6
4
2
0
0
10
20
30
40
50
60
70
80
90
100
Number of Iterations


Value of Norm of Gradient vs. Number of Iterations, A = 1, Gradient Descent
Value of the norm of the gradient
Number of Iterations
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
-10
0
10
20
30
40
50
60
70
80
90

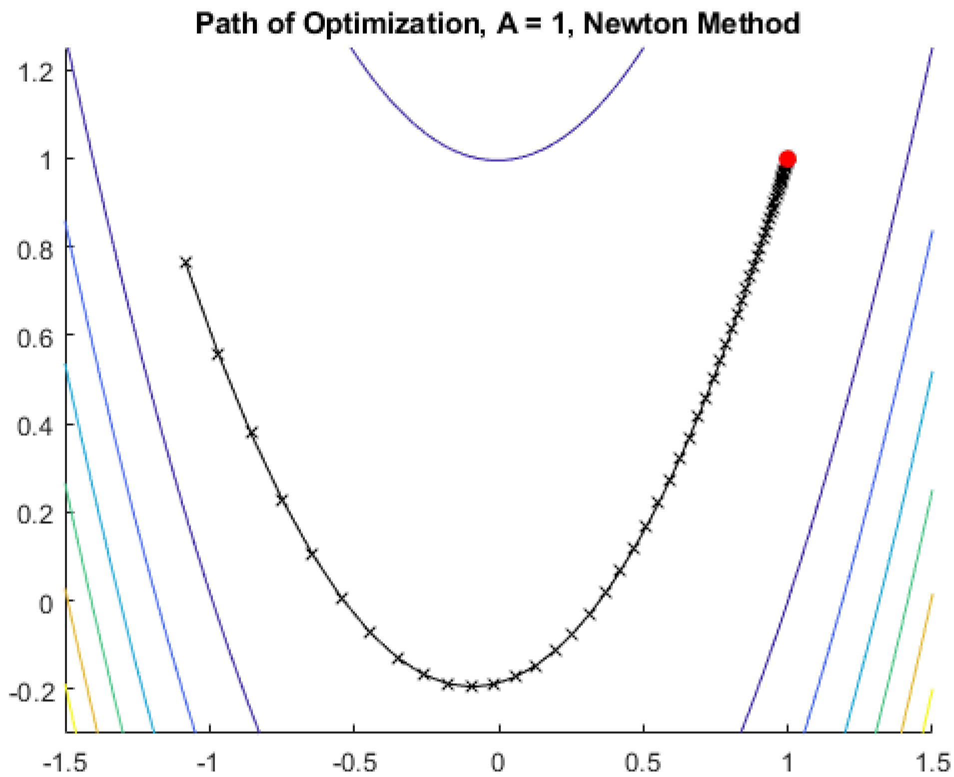

Path of Optimization, A = 1, Gradient Descent
1.2
1.1
1
0.9
0.8
0.7
0
0.5
1
1.5

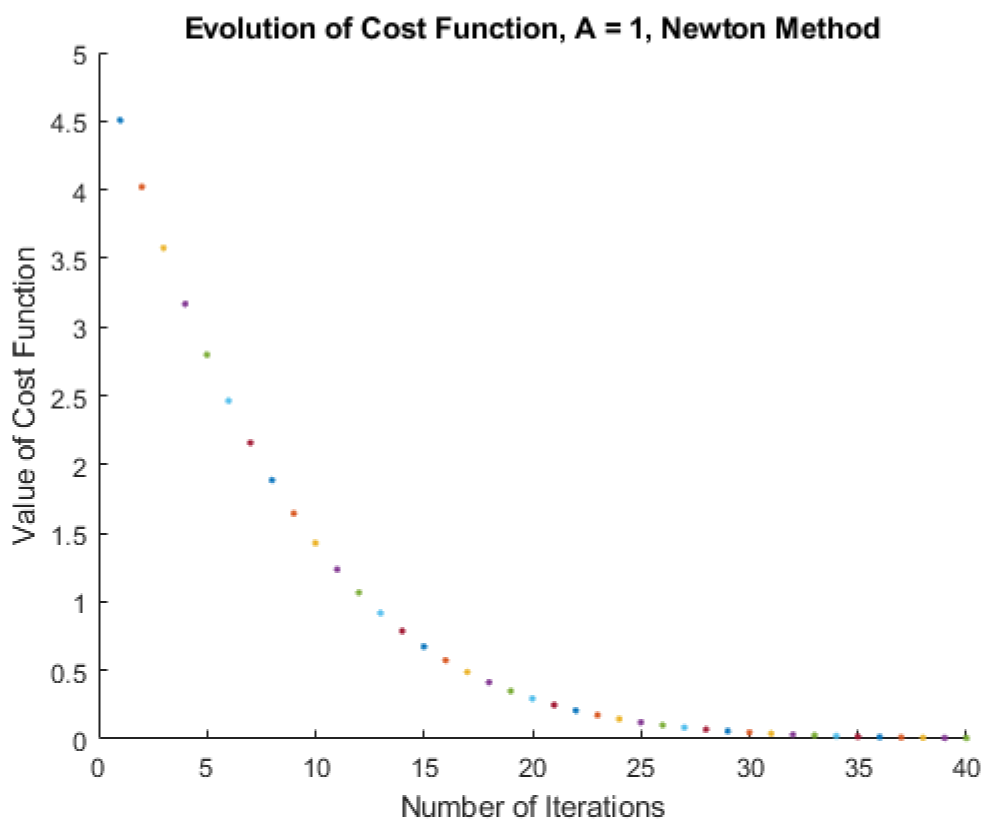


Evolution of Cost Function, A = 1, Gradient Descent
Value of Cost Function
Number of Iterations



Value of Norm of Gradient vs. Number of Iterations, A = 1, Newton Method
Value of the norm of the gradient
Number of Iterations


Path of Optimization, A = 1, Newton Method
1.2
1
0.8
0.6
0.4
0.2
0
-0.2
-1.5
-1
-0.5
0
0.5
1
1.5


Evolution of Cost Function, A = 1, Newton Method
Value of Cost Function
Number of Iterations
5
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
0
5
10
15
20
25
30
35
40

Published with MATLAB® R2022b